1. Лабораторна робота №1

Тема: "Процеси та потоки"

Взаємодія між процесами.
Розподіл даних між процесами.
Робота з файлами які відображуються у пам'ять.

Завдання 1

Одна програма буде сортувати дані у файлі, а інша відображати вміст цього файлу. Працювати обидва процеси будуть одночасно. **Третя програма** буде створювати (або заповнювати по новому) масив випадкових чисел.

Програма №1. "Сортування даних" (консольна)

Беремо за основу програму "Hello windows"
Включаємо обробку події натискання клавіші, і відстежуємо в ньому натискання пробілу. Якщо користувач натиснув пробіл, значить починаємо сортування даних. Виконуємо проектування файлу в пам'ять. Використовуємо для цього створений файл data.dat. В результаті отримаємо доступ до даних як до звичайного одновимірного масиву.
Виконуємо сортування масиву, будь-яким з методів сортування. Вставте 1-но секундну затримку для кожної ітерації сортування масиву, це дозволить потім наочніше побачити процес сортування.
По закінченню сортування, програма виводить у вікно, рядок «Робота завершена».

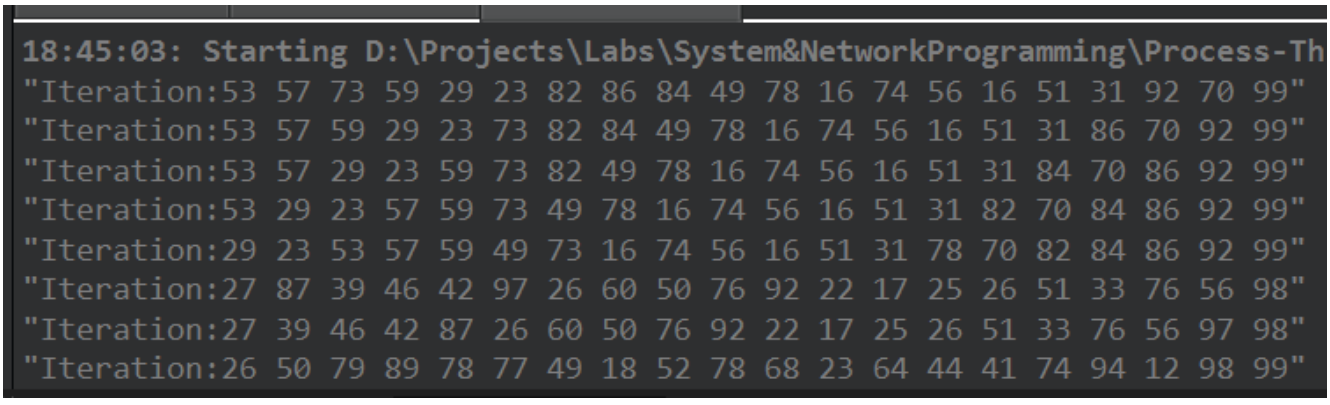

Рис. 1.1. консольна программа для сортування в зворотньому напрямку

Програма №2. «Виведення файлу даних у вікно» (віконна)

Виконуємо проектування файлу в пам'ять. Використовуємо для цього створений файл data.dat. В результаті отримаємо доступ до даних як до звичайного одновимірного масиву. Цей же файл проектує в пам'ять попередня програма. Створюємо таймер на 0.5 секунди. При отриманні повідомлення від таймера, виконуємо висновок всього масиву в вікно. Передбачте коректний перевивід даних у вікно, без накладень. У вікно виводиться не числа з масиву, а рядки одного і того ж символу, наприклад «*», в кількості, що дорівнює числу з масиву.

Запускаємо на виконання обидві програми одночасно. Коли друга програма запустилася і виконує висновок даних у вікно (виводить поки одну й ту ж саму картинку кожні пів секунди), натискаємо пробіл в першій програмі і вона починає сортувати масив. При цьому, так як вони дані беруть з одного і того ж файлу (обидві проектували його собі на згадку), то перша вносить зміни переставляючи дані при сортуванні, а друга виводить з себе у вікно і ми бачимо хід процесу сортування. Тимчасову затримку в першій програмі можна при потребі збільшити.

Ці дві програми демонструють можливість організації спільного доступу процесів до одних і тих самих даних. Так само демонструється механізм проектування файлу в пам'ять, як один з найкращих методів доступу до файлу.

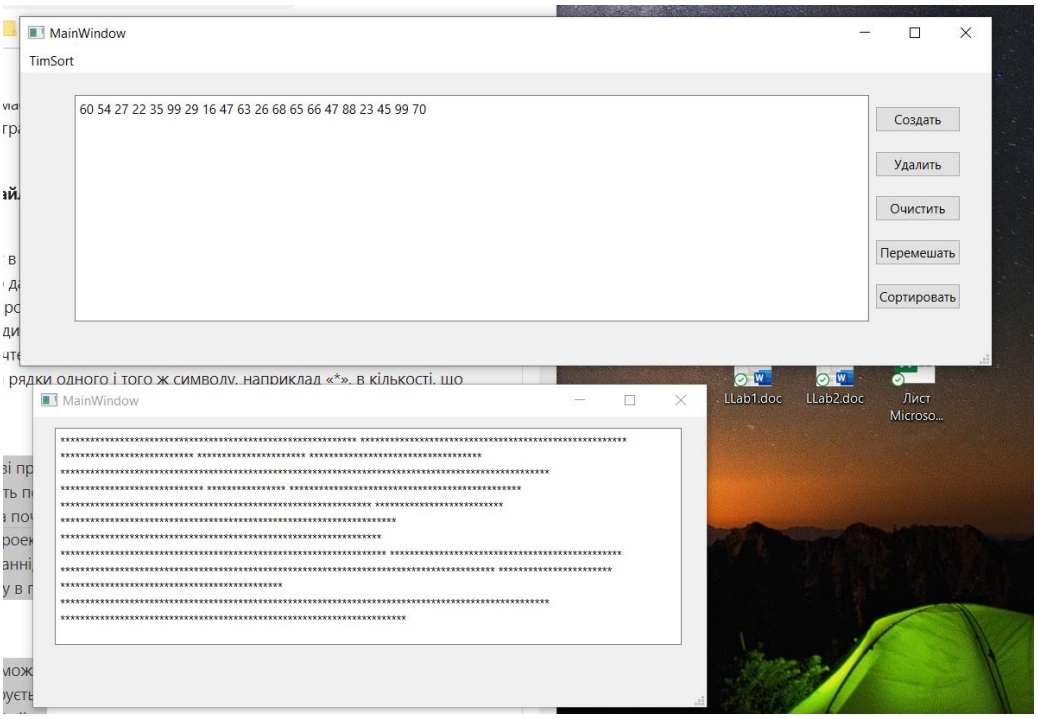

Рис. 1.2. Ці дві програми демонструють можливість організації спільного доступу процесів до одних і тих самих даних.

Завдання 2.

Для коректної роботи зі спільними даними у цих двох програмах потрібно додати **синхронізацію потоків**, які можуть одночасно звертатися до спільних даних.

```
QMutex mutex; // Объект мьютекса для синхронизации доступа к разделяемой памяти

QTimer timer;

QObject::connect(&timer, &QTimer::timeout, [&](){
    QString res = "Iteration:";
    bool swapped = false;

    {
        QMutexLocker locker(&mutex); // Блокируем мьютекс при доступе к данным
```

Рис. 1.3. синхронізація в першій програмі при сортуванні

Для організації такої синхронізації потрібно використати об'єкт ядра ОС **mutex або semaphor**, або інший синхронізуючий об'єкт, а також **функції очікування** (наприклад, **WaitForSinglJbject()**).

Також обов'язковим є використання **обробки виняткових ситуацій** в роботі вище описаних трьох програм. Бо, некоректна робота будь якої з трьох, викличе неправильну роботу інших, через блокування спільних даних.

```
    try {
        if (!sharedMemory.attach()) {
            if (!sharedMemory.create(sizeof(int) * 20)) {
                throw std::runtime_error("Failed to create or attach to Memory Mapped File");
            }
        }
        updateSharedMemoryData();

    } catch (const std::exception& e) {
        QMessageBox::critical(this, "Error", e.what());
    }
}

void MainWindow::updateSharedMemoryData()
{
    if (sharedMemory.isAttached())
    {
        QMutexLocker locker(mutex);
        int *data = static_cast<int *>(sharedMemory.data());

        for (int i = 0; i < dataVector.size(); ++i)
        {
            data[i] = dataVector[i];
        }

        QStringList dataStringList;
        for (int i = 0; i < dataVector.size(); ++i)
        {
            dataStringList.append(QString::number(dataVector[i]));
        }

        ui->textBrowser->setPlainText(dataStringList.join(" "));
    }
}
```

Рис. 1.3. синхронізація в другій програмі при будь-яких маніпуляціях з спільними даними

```cpp
}

void MainWindow::updateData()
{
    QMutexLocker locker(mutex);

    if (sharedMemory.isAttached()) {
        int *data = reinterpret_cast<int *>(sharedMemory.data());

        QString dataString;
        for (int i = 0; i < 20; ++i) {
            dataString += QString("*").repeated(data[i]) + " ";
        }

        if (dataString.isEmpty()) {
            ui->textBrowser->setPlainText("Данные отсутствуют.");
        } else {
            ui->textBrowser->setPlainText(dataString);
        }
    }
}
```

Рис. 1.4. синхронізація в третій програмі при виведенні даних

Для обробки виняткових ситуацій, необхідно правильно визначити **критичні секції коду** усіх написаних програм.

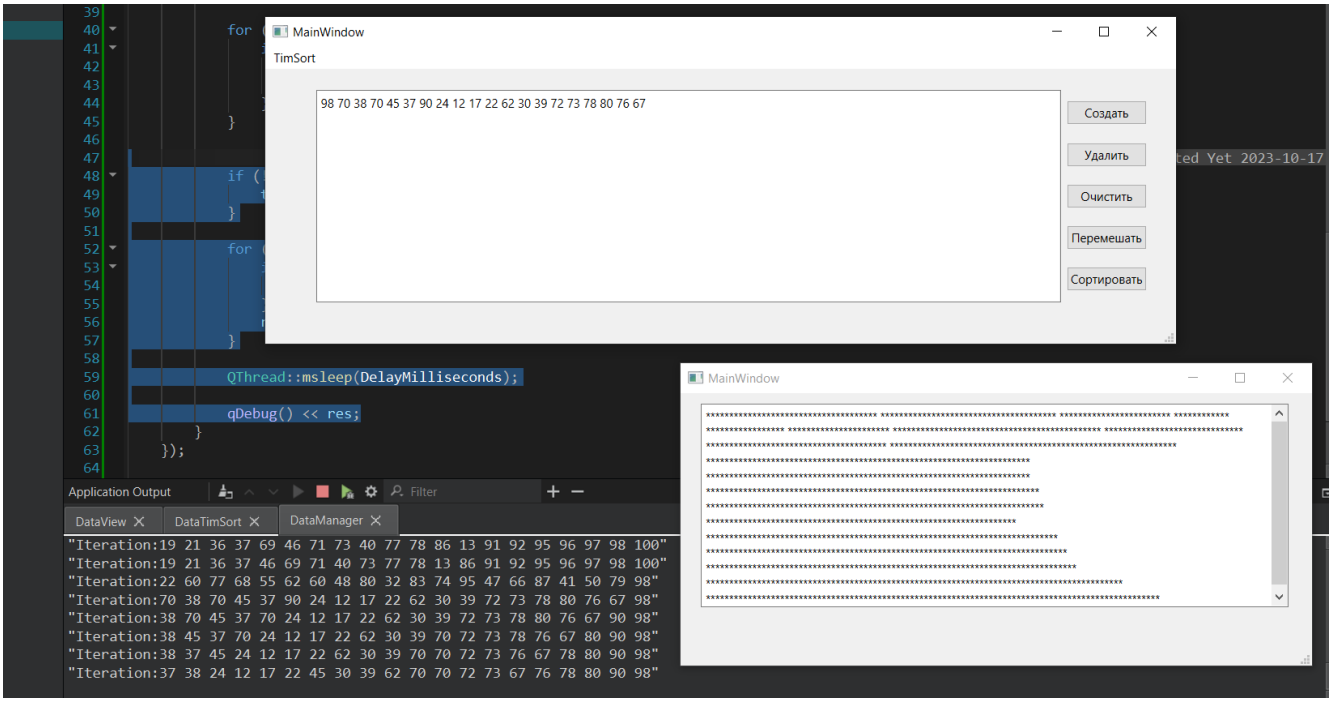

Рис. 1.5. приклад роботи трьох програм з однією областю пам’яті (без конфліктів), можна одночасно сортувати створювати, перемішувати чи сортувати

Однією з можливих помилок є спроба розблокувати або заблокувати м'ютекс під час виконання іншої операції, коли м'ютекс на даний момент не заблокований або, навпаки, вже заблокований. Це може призвести до виникнення такої помилки:

```
ASSERT: "!m_isLocked" in file D:/Utilites/Qt/6.5.2/mingw_64/include/QtCore/qmutex.h, line 267
```

Ця помилка вказує на спробу виконати некоректну операцію з м'ютексом, коли його поточний стан не відповідає очікуваному.

І программа вісне:

Під час синхронізації доступу до файлу в пам'яті можуть виникнути різні помилки та проблеми, ось декілька з них з прикладами:

1. Deadlock (взаємна блокування): Ця ситуація виникає, коли кілька потоків або процесів блокують один одного та чекають на ресурс, який утримує інший. Наприклад, поток A блокує ресурс X та чекає на ресурс Y, в той час як поток B блокує ресурс Y та чекає на ресурс X.

2. Race Condition (ситуація гонки): Це виникає, коли два або більше потоки або процеси намагаються змінити спільний ресурс без синхронізації. Це може призвести до непередбачуваного стану даних.

3. Швидкість доступу до ресурсу: Якщо один потік займає ресурс на довгий час, інші потоки можуть бути блоковані, чекаючи на доступ до цього ресурсу. Це може призвести до низької продуктивності та затримок.

4. Неправильний порядок блокування: Якщо потоки блокують ресурси в різному порядку, це може призвести до deadlock або неправильної поведінки програми.

Але всі проблеми було усунуто.

Висновки:

1. У даній лабораторній роботі досліджувалася взаємодія між процесами та робота з файлами, які відображуються в пам'ять.
2. Були створені три програми для взаємодії між процесами:

a. Перша програма виконує сортування даних у файлі.
b. Друга програма відображає дані з цього файлу в вікно і виводить їх на екран кожні 0.5 секунди.
c. Третя програма створює або заповнює масив випадковими числами.

3. Для коректної роботи зі спільними даними в програмах була додана синхронізація потоків за допомогою мютексів.
4. Важливим аспектом, який також був доданий - є обробка виняткових ситуацій для запобігання блокуванню спільних даних.
5. У лабораторній роботі демонструється можливість організації спільного доступу до даних в одній області пам'яті і взаємодії між програмами через спільний ресурс.

Отже, ця лабораторна робота відображає важливі аспекти взаємодії між процесами, синхронізації та спільного доступу до ресурсів.